The Book of Spells

The Spellcaster™ Language Manual

Shenandoah Software

Alphabetical Listing of Spellcaster Syllables

A	AKA	change pen color to next higher color	15
B	BO	step forward, draw (click)	11
C		unused	
D	DI	turn around	14
E	E	EA to EZ, E' expose teleporter (remember position)	52
F	FE	FEA to FEZ, FE' felt teleporter (go there)	52
G	GO	wash (click), step forward, wash (click)	12
H	HI	turn up (clockwise in border)	14
I	IX	nothing if unmatched, exit if matched	44
J	JO	step forward without drawing or clicking	11
K	K	KO, KA, KE, KI, KU, KX set pen color	38
L	LI	turn left (does nothing in border)	14
M	M	MO, MA, ME, MI, MU, MX repeat so many times	34
N	NU	repeat	36
O	OX	nothing if matched, exit if unmatched	44
P	PU	left parenthesis (down one level)	29
Q	QUSH	turn clicks on/off	15
R	RI	turn right (does nothing in border)	14
S	SPELLS	gives a spell a name, remember in memory	26
T	TU	begin spell name	28
U		unused	
V	VUZIM	check keyboard for a one-key spell name	31
W	WU	call keyboard for a spell	30
X	XEN	nothing if IX or OX caused exit in prior word, else exit	46
Y	Y	flip left and right (mirror image)	15
Z	ZIM	right parenthesis, ends PU, TU, or WU	28

Quickfind

Some Useful Spells Completely Explained 75
SNIF 76
The PARabase spells 76
The FILL spell 80
Spellcaster Syntax 81
Index 93

this belongs to
Scott Gilkeson

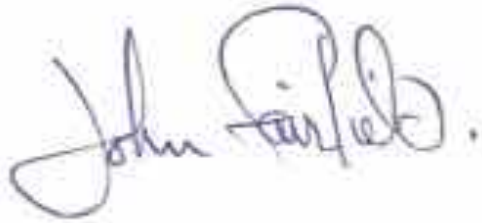
The Book of Spells

The Spellcaster Language Manual

Version 1

by
John R. Fairfield

DON'T READ THIS BOOK until you've played with Spellcaster. This book won't make much sense until you've seen Spellcaster in action, and read some of the tutorial that is built into the Spellcaster disk. Spellcaster will tell you when you are ready for The Book of Spells.



What is Spellcaster?

Spellcaster is a programming language that was designed to be very easy to learn, yet be powerful enough for programming fast video games. Small children enjoy Spellcaster because it lets them draw colored patterns all over the screen to their heart's content, just by pushing on a few keys on the keyboard. Spellcaster quickly pulls the beginner into serious computer programming.

Spellcaster has many unique features that make it an ideal tool for learning to program:

* Feedback. Spellcaster makes all the program's data visible all the time, not as numbers, but as screen patterns. The effects of every keystroke are instantly shown on the screen. Every aspect of the program speaks to you: it makes noises, you can see its progress on the screen, its conditional branching test is continuously on display; in fact, nearly every aspect of the entire current state of the program is continuously on display.

* Editing. Playing with programs themselves is so easy in Spellcaster. You can stop a program, make it back up, change it, add stuff to it, all while watching the instant effects of every change.

* On-screen Tutorial. A tutorial (dozens of lessons) is built into the Spellcaster disk, to teach the beginner how to use Spellcaster. It displays text on the screen, tells the beginner what to type next, even does some typing for the beginner. The tutorial uses simple video games as example programs, to teach beginners, step by step, how to program.

* Simplicity. For instance, there are only three kinds of conditional test in Spellcaster, and in the whole language there are only 34 syllables.

What is a spell?

A spell is a computer program written in the Spellcaster language.

My betters always told me that a spell was a long stream of gibberish syllables. Some wild-eyed wizard would utter the spell, and the door would blast open. At least that's what we learned in the Army.

In Spellcaster, a spell is a long stream of syllables that, as you type them, cause all sorts of things to happen on the screen. Each syllable of the spell makes some simple change to the screen, but the combined effect of the whole spell can be very, very interesting.

Can I really learn computer programming with Spellcaster?

Is Spellcaster just for games or is it serious? There are a number of other systems that let you draw stuff on the screen, and that are advertised as introductions to computer programming, but they don't measure up to computer science's definition of true computer languages. (An exception is LOGO, which, like Spellcaster, is a true computer language.) With Spellcaster, will you really learn computer programming? Yes. Spellcaster is a true, complete formal language.

Let's be specific. If you think that computing necessarily has to do with numbers, then you probably wonder why Spellcaster doesn't deal with numbers. But the heart of computing isn't numbers at all, it's logic. Most introductory courses in BASIC (a popular computing language), spend so much time dealing with numbers that they never get into interesting control logic. Spellcaster takes you directly to the heart of computing, to the logical control of a changing process. Logic isn't easy, and if you want to write really complicated programs, it won't be easy. But if you use Spellcaster, at least there will be no barriers between you and the truly interesting, truly difficult part of computer science.

To give an example: several systems which only let you draw stuff on the screen (without really getting you into programming) have a special FILL command that lets you fill in a region with a solid color. You might use the FILL command to color in the roof of a house, after you drew the outline of the roof. There is no such commmand in Spellcaster. Why? Because you can write a spell which will do that, a spell that will fill in any outlined region you make, with any color you want. Now the FILL spell is already written, it's on your Spellcaster disk, the tutorial will tell you all about it and

how to cast it, and this manual, in the last section, explains the FILL spell for you in great detail. So you get the best of both worlds: since the spell is already written, you can use it just as easily as any special FILL command, and you can also learn from it, change it, and adapt it to your own spells.

Why Spellcaster doesn't use English

Many computer languages (like LOGO or BASIC) use "English-like" phrases for their commands. The question is whether using English phrases is a help or a hindrance. Usually the word as used in the computer language doesn't quite mean the same thing as it does in English, and that can be very confusing. What's even more confusing for the learner is when you can use some English phrases but not others that appear equally reasonable.

For example, in BASIC, "FOR I = 51 TO 60" is supposed to mean that something is to be repeated 10 times, once with I (a number) equal to 51, the next time with I equal to 52, and so on. But "FROM I = 51 TO 60" means nothing in BASIC, nor does "FOR I EQUALS 51 TO 60", nor does "FOR I FROM 51 TO 60", nor does...

Spellcaster does not confuse itself with English.

Computer literacy concepts in Spellcaster

+ reading technical manuals
+ the program/data dichotomy
+ the difference between computer memory and secondary (disk) memory.
+ editing, cursor control, control keys
+ creating, deleting, changing files on a disk
+ variables
+ interpreters
+ powers of 2 arithmetic
+ structured programming control structures, including
    loops
    loop exit conditions
    nested conditions
    subroutines
    recursion
+ stacks
+ strings
+ process, process state
+ polling
+ real time programming
+ random number generators

how to read it, and this manual, in the last section, explains the FILL spell for you in great detail. So you get the best of both worlds: since the spell is already written, you can use it just as easily as any special FILL command, and you can also learn from it, change it, and adapt it to your own spells.

Why Spellcaster doesn't use English

Many computer languages (like LOGO or BASIC) use "English-like" phrases for their commands. The question is whether using English phrases is a help or a hindrance. Usually the word as used in the computer language doesn't quite mean the same thing as it does in English, and that can be very confusing. What's even more confusing for the learner is when you can use some English phrases but not others that appear equally reasonable.

For example, in BASIC, "FOR I = 51 TO 60" is supposed to mean that something is to be repeated 10 times, once with I (a number) equal to 51, the next time with I equal to 52, and so on. But "FROM I = 51 TO 60" means nothing in BASIC, nor does "FOR I EQUALS 51 TO 60", nor does "FOR I FROM 51 TO 60", nor does...

Spellcaster does not confuse itself with English.

Drawing on the Screen

Spellcaster treats the screen as though it were a grid (a checkerboard) of little squares. At first, all these squares are colored black. A spell can color in the squares. It's as if the computer had a pen, and started stepping from one square to the next, drawing as it went.

The Main Screen and the Border

Actually, Spellcaster thinks that there are two screens. One is big, and fills most of the TV's (or monitor's) screen. This big screen is called the "main screen". The other screen is just a thin line like a picture frame all the way around the main screen. This frame screen is called the "border".

In the main screen, if BO or JO or GO makes the pen go off one side of the main screen (as if it were trying to step into the border), the pen immediately comes back in on the other side (without ever getting into the border). Same way for top and bottom: if you step out the top, you'll come right in at the bottom.

The border is like a railroad track one square thick that goes all around the main screen. If the pen is in the border (in other words, on the railroad track) and comes to a corner of the border, going straight ahead with BO, JO or GO makes the pen follow the track around the corner. LI and RI don't work in the border: you can't turn sideways to the track at all. To turn around in the border you have to use DI.

There are really only two directions in the border, clockwise and counterclockwise around the track. In the border, RI makes the pen turn to the clockwise direction around the border, no matter what direction it had been going. So there's no way that you can step out of one screen into the other with BO, GO or JO.

To get from the main screen to the border, or back, you have to use Teleporters.

BO

BO is useful for drawing and for making sounds.

BO makes the pen step one step forward, and color in the square it steps into. It also usually makes the computer's speaker give a little click, though the sound can be turned off (quickfind the Menu, and the syllable (ESSH). When a spell does a bunch of BO's in a hurry, the clicks are so fast they sound like a continuous tone.

JO

JO is useful for changing your position without drawing anything.

JO makes the pen step one step forward, but it doesn't color any square, and it makes no click.

GO

GO is usually used to move a dot one step farther on, by erasing the dot in the old square and drawing a new dot next to it in the new square. But GO doesn't really "erase" or "draw", it "washes".

GO makes the pen step one step forward, and it uses the pen color to "wash" both the square it's leaving and the square it arrives on. When GO "washes" a square with the pen color, the square usually changes color, but the new color isn't necessarily the pen color. What exactly do I mean by "wash"? Well, suppose X and Y are two colors (one of them is the color of the square getting washed, and the other is the pen color). Then these are the three laws of washing:

1. X washed with Y makes the same color as Y washed with X.

(For instance, if you wash a green square with orange pen color, you get the same color as if you washed an orange square with green pen color.)

2. X washed with X0 (the zero color) makes X.

(That means that washing a square when the pen color is X0 doesn't change the square. 1 and 2 together mean that washing a X0 colored square with pen color X makes the square X colored.)

3. If you wash a square twice in a row with the same color, the second time "undoes" the effect of the first.
In other words,
If X washed with Y makes color Z,
then Z washed with Y makes X.

LI RI

In the main screen, LI makes the pen turn to its left (though it doesn't take any steps in that direction yet), and RI makes the pen turn to its right. Three LI's together do the same thing as one RI (and three RI's do the same as one LI). Two LI's turn the pen around, just like two RI's.

LI and RI have no effect in the border. If you want to turn around in the border, you have to use DI.

DI

DI makes the pen turn around (change direction).

HI

In the main screen, HI makes the pen turn up, no matter what direction it had been going. In the border, HI makes the pen head clockwise, no matter what direction it had been going.

Y

Y is useful for making symmetrical shapes, where each side is the mirror image of the other.

Y turns left and right, so that now L1 makes the pen turn right, and R1 makes the pen turn left. If you cast a spell, and then type Y, and then cast the spell again (example: TUBIL1MYTUBIL1M), the second time will be the mirror image of the first.

QUIM

QUIM is useful if you want parts of a spell not to make any noise.

QUIM turns the clicking made by B1 and QQ off. A second QUIM turns the clicks back on again, a third turns them off, a fourth back on, and so on. If you've turned off the clicks with the menu though, they're off for good, or until you turn them back on with the menu.

AKA

AKA changes the color of the pen to the next higher pen color. When the highest color is reached, AKA sets the pen color back to the lowest color. You can always tell what color the pen is by looking at the top square in the matchbox.

Special Keys

There are a few keys that have special effects when you are using Spellcaster. These special keys are [(ESCAPE), ?, ;, :, [(BACKSPACE)], \, left arrow, right arrow, -, +, and *.

Activity Change Keys

When you are using [illegible], you're usually doing one of three activities:

1. You're looking at the drawing screen, casting spells.
2. You're looking at the command menu, choosing a command.
3. You're looking at a page (for example, of the tutorial).

The following keys take you from one activity to another.

[illegible] RETURN): Go to the drawing screen (and clear the drawing screen).

[illegible]: Go to the command menu.

[illegible]: Go to the previous page.

[illegible]: Go to the next page.

The Keystroke Stack

The best way to learn about the keystroke stack is to type a small spell, and then play with these keys: delete ([illegible]), rightdelete ([illegible]), leftarrow, rightarrow, superleftarrow ([illegible]), superrightarrow ([illegible]), and supperleftarrow (*). You can think of the keystroke stack as an invisible bag where the Spellmaster system keeps a pile of keystrokes. Sometimes keystrokes are put into the bag, sometimes they are taken out. The last keystroke put in will be the first one taken out.

Hints

The Spellmaster tutorial can put keystrokes into the keystroke stack. When it does this, you can "type" the keystrokes in the stack just by typing right arrow, which removes the keystrokes one by one from the stack and types them for you. In this way the tutorial can demonstrate how to type all kinds of things in Spellmaster.

Using the Keystroke Stack

The following series of examples might help you to understand how the keystroke stack works. Suppose you have typed the keystrokes "SBQABRB", which show up as

?USBQQBRAABQRIBD

Then you you hit the leftarrow key four times. The last four keystrokes you typed (ABRB) will be eaten up. The A will be flashing at you from the cursor, the rest (BRB) will be hidden. I will draw the picture like this, to show you the hidden keys that are squirreled away in the keystroke stack:

←——the most recent keystroke in the stack blinks.

←——the first keystroke that was put in the stack is buried 4 deep.

Now, here are some pictures that show what happens after you hit one of these special keys, starting each time with the picture on the bottom of the opposite page:

After Delete (/)

After Backdelete (\)

After Leftarrow (←)

After Rightarrow (→)

After Superleftarrow (↑)

After Superrightarrow (↑)

Delete (/)

Delete removes the last keystroke from the spell you are typing, and redoes the spell up to that point.

Rightdelete (\)

The rightdelete key removes the topmost keystroke (the one that blinks at you) from the keystroke stack.

Leftarrow (←)

Leftarrow is like delete, only it puts the deleted keystroke onto the top of the keystroke stack. In this way, the keystroke stack can remember the deleted keystrokes, so that it can cough them up for you (when you use rightarrow).

Rightarrow (→)

Rightarrow removes the top keystroke (the one that blinks at you) from the keystroke stack, and types that key for you.

Superleftarrow (-)

Superleftarrow has the same effect as if you beat on the leftarrow key until the entire spell were put into the keystroke stack.

Superrightarrow (+)

Superrightarrow has the same effect as if you beat on the rightarrow key until all the keys in the keystroke stack were used up. The superrightarrow key makes Spellcaster start getting keystrokes from the keystroke stack, until the stack is empty, or until you hit superrightarrow again. If you hit superrightarrow while the spell is gobbling keystrokes from the keystroke stack, it will stop. This is handy if you've got a long spell in the keystroke stack, and you want to go to halfway through in a hurry: hit superrightarrow twice, fast, and see how far it got before you stopped it.

Zapperleftarrow (*)

Zapperleftarrow does the same thing as superleftarrow, but it also makes Spellcaster turn off spellcasting. You can still type in spells, but nothing happens on the drawing screen. This is especially handy if you are trying to change a spell that takes a long time to run; you can work much faster if you turn off spellcasting. If spellcasting is turned off and you hit the zapperleftarrow key again, it turns spellcasting back on.

Defining and Casting Spells

A spell is made of words. Words are made of syllables.

Example syllables: BO WU IX MU KA. A syllable is few letters long, and you usually just have to type the first letter to get the whole syllable.

Example words: BORIBOLI XENBO NUBOIX. A word is made of one or more syllables, and ends with a blank (though there's one way of enclosing blanks within a word: quickfind PU).

A spell is made of one or more words. Every time you want to cast a spell, do you have to type the whole thing? No, you can give a spell a name, and make the computer remember the spell and its name. Then, every time you type the name of the spell, it happens just as if you had retyped the whole thing.

The next page tells you how to give a spell a name.

SPELLS

To give a spell a name, follow these steps:

1. Type the spell.

   Example: NUBORIBILIIX RIBOBOBO

2. Hit the S key (which stands for the syllable "SPELLS").
   (The computer will print "SPELLS TU" and wait for you
   to type any name you want.The TU is a syllable that
   marks the beginning of a spell name: all spell names
   begin with TU.)

   Example: NUBORIBOLIIX RIBOBOBO SPELLS TU

3. Type a name for your spell.

   Example: NUBORIBOLIIX RIBOBOBO SPELLS TUWALLPAPER

4. Hit the Z key (which stands for the syllable ZIM).
   (All spell names end with ZIM.)

   Example: NUBORIBOLIIX RIBOBOBO SPELLS TUWALLPAPERZIM !

One occasional hassle is that you think of a name that has a Z in it (like ELIZABETH), but then find out you can't use it, because as soon as you type the Z, the computer thinks it's for the closing ZIM.

Suppose you typed no name at all, just hit Z right off, what happens? Well, that's the name that has no letters in it, and it's a perfectly good name. (To cast that spell you would type TUZIM.)

Can you put spaces and numbers in a spell name (like TU96THZIM or TUMY SPELLZIM)? Yes. Most of the keys on your keyboard can be used in spell names. It changes some from computer to computer, but at least all of the letters (except Z), the digits (01234567891, and blanks (the space bar) can always be used in spell names.

If the computer already has in its memory another spell that has the same name as the one you chose, it will complain by printing the message "There's already a spell with that name", and make you make up a different name.

The computer will remember the spell until you turn it off, or otherwise quit Spellcaster. If you want your spells to be written onto the disk so that they can be used when you turn the computer back on, you have to tell the computer to do that. (Use the Menu).

TU...ZIM PU...ZIM WU...ZIM VUZIM

All of these syllables cause one spell to cast another spell. When the second spell is done, the first spell continues on with the rest of itself (the part coming after the ZIM). For example, BOBOBOTUCOWZIMJOJOJO does three BO's, then all the syllables in the COW spell, then three JO's.

TU...ZIM

To cast a spell, you type its name between TU and ZIM. For example, you cast a spell named COW by typing TUCOWZIM. The TU tells the computer to look up the spell named COW. If the computer doesn't know any spell named COW, it prints "There is no spell named COW." Maybe you haven't named any spell COW yet. Maybe you misspelled the name. Or maybe you've forgotten to have the computer read your spells (that you made and named yesterday) from the disk (quickfind IN).

You can make a spell that casts a spell that doesn't exist: the computer will just print that message ("There is no spell named ...") every time you cast the spell, and continue as if the missing spell did nothing. (Why? Quickfind Recursion)

PU...ZIM

PU and ZIM are useful when you want to cast a short spell inside another spell, and you don't want the bother of giving the short spell a name. PU and ZIM are parentheses (these () things). They let you put a spell in the middle of another spell (just like these parentheses let me put this sentence in the middle of another sentence) without having to name the spell you put in. For example, suppose you want to make a box of four (MEMO) sides. Each side is made of fifteen (MIMI) BO's. (It's OK if you don't understand MEMO and MIMI yet, just remember for now that MEMO means 4 and MIMI means 15.) You could do it by first making a spell with 15 BO's:

```
MIMIBO SPELLS TUSIDEZIM !
```

and then doing four sides like this:

```
MEMOTUSIDEZIMRI
```

Or, you could do it with just one spell, like this:

```
MEMOPU MIMIBO ZIMRI
```

In both cases, the box spell behaves like MEMO(MIMIBO)RI. Whatever is between the PU and the ZIM is treated exactly like a regular spell (which of course means that you could have parentheses (like this (or even this)) inside parentheses, and so on). Notice that if you type

```
MEMOMIMIBORI
```

without the PU and ZIM, you just get BORI repeated MEMOMIMI times (that's 143 times, and it doesn't make a box).

WU...ZIM

Suppose you have a spell that goes along and makes some regular pattern. It would be nice if every once in a while it would come up for air and ask you for some redirection. If your spell has a WU in it, then every time it gets to the WU it will wait for you to type a spell at the keyboard. If you want it to continue with no change, just hit ZIM. But if you want it to hang a right, or if you want to cast a spell at that point that draws a flower, go ahead. Then hit ZIM, and the original spell will go on its way.

Every time a spell does a WU, it asks you to type in a spell to be cast at that point. The cursor (the bright thing that shows where your typing will appear) appears one line down, telling you that the computer is waiting for you to type. You can type anything you would normally put in a spell (as long or short as you want). When you are done typing the spell, type ZIM. Then the spell that had the WU in it will continue.

For various reasons, a WU disables delete, leftarrow, superleftarrow, and zapperleftarrow. In other words, if a WU asks you for a spell, you can't back up until you hit ZIM.

VUZIM

VUZIM is good for making game spells, where you want to control some process (like a rocket) that is going to keep going (smack into an astroid) even if you don't do anything to control it (in fact, especially if you don't do anything to control it). Every time a spell does a VUZIM, it checks, real quick, to see if you have hit a key on the keyboard. If you haven't, the spell continues as if the VUZIM wasn't even there (moving the rocket ever closer to that astroid).

If you have hit a key, the computer looks to see if there is a spell that has that name (like if you hit the A key, the computer looks for a spell named A). If there is no such spell, the spell with the VUZIM continues as if nothing had happened. But if there is, that spell is cast, right then, before the first spell continues another step (hopefully, TUAZIM will turn that poor rocket around).

Because VUZIM doesn't stop the spell it's in, it is useful for games where the game keeps going even if you don't type anything. WU, by contrast, stops the spell it is in, to wait on the keyboard.

VUZIM can only cast spells whose names have just one letter in them. With VUZIM you can use any of the keys (except the Special keys) to cast spells, except, one might think, the Z key. No spell can be named Z, because typing Z ends a spell name (with a ZIM). However, if VUZIM spots that you've pressed the Z key, it looks for the spell whose name has no letters in it, the spell that would be cast as TUZIM. If this spell exists, VUZIM will cast it when you hit Z.

Repeaters

Repeaters are useful if you have some pattern that you want repeated a number of times on the screen.

Repeaters are prefix syllables. A prefix is a syllable that is tacked on to the beginning of a word, to change the meaning of the word. For example, "im" changes the word "possible" to "impossible".

A repeater prefix makes the rest of the word repeat a certain number of times (unless an IX or QX exits the word). For example, the prefix MI means "repeat three times", so MIBORIBOLI does BORIBOLI three times. Now the spell BO RI BO LI usually has the same effect as the spell BORIBOLI. It usually doesn't matter whether the spell is four separate words (BO RI BO LI) or just one word (BORIBOLI). But if you put a repeater in front, the spaces between the words become important. MIBO RI BO LI does three BO's, then one RI, one BO, and one LI.

But MIPU BO RI BO LI ZIM does the same thing as MIBORIBOLI. Why? Because the PU...ZIM parentheses wrap the BO RI BO LI up into a spell. It's just as if you typed

BO RI BO LI SPELLS TUSTEPZIM !

and then did MITUSTEPZIM. You see, because the spaces were between the words of another spell, the word that started with MI didn't stop until after the ZIM.

MO MA ME MI

Make sure you see the IMPORTANT POINT at the bottom of the next page. Now, suppose you want something to repeat 11 times. What repeater to use? This table gives you all the repeaters between 0 and 15.

MO: 0	MA: 1	ME: 2	MI: 3
MAMO: 4	MAMA: 5	MAME: 6	MAMI: 7
MEMO: 8	MEMA: 9	MEME: 10	MEMI: 11
MIMO: 12	MIMA: 13	MIME: 14	MIMI: 15

The whole thing is built on four syllables: MO, MA, ME and MI which are worth 0, 1, 2 and 3.

See how the repeaters in the first column (0, 4, 8 and 12) all end with MO. The second column repeaters all end with MA, the third column with ME and the fourth column with MI.

In effect, all the repeaters in the first row begin with MO, because MOMO is the same as MO, MOMA is the same as MA, MOME is the same as ME, and MOMI is the same as MI. Why? Because MO means zero, and so doesn't add anything. All the repeaters in the second row begin with MA, and all in the third row begin with ME, and all in the last row begin with MI.

Suppose somebody asks you how much MIMI is (don't look!). How could you figure it out? Well, all you have to know is that MO is zero, MA is one, ME is two, and MI is three. That's all. Then what you do is take the first syllable of MIMI, multiply it by 4, and add the last syllable. That's MI times 4, plus ME. Which is three times 4, plus two. Which is 12+2 = 14. Now look.

If a repeater has three syllables, like MIMAMO, then the first syllable is multipled times 4 twice, the second syllable times 4 once, and the last syllable just added on. So MIMAMO is 3*4*4 + 1*4 + 0 = 52.

If a repeater has four syllables, like MEMOMEMA, then the first syllable is multiplied times 4 three times, the second syllable times 4 twice, the third syllable times 4 once, and the last syllable just added on. So MEMOMEMA is 2*4*4*4 + 0*4*4 + 2*4 + 1 = 137.

On some computers, the biggest repeater you can make is MIMIMIMI. That's 3*4*4*4 + 3*4*4 + 1*4 + 3 = 255. If you try to make repeaters that have more than four syllables, they start over again at zero. Like MAMOMOMOME is worth just ME, 2.

*** IMPORTANT POINT ***: Being able to figure out repeaters isn't especially important for using Spellcaster. If you ask me (John Fairfield, the author of Spellcaster) how much MEMAMO is, I don't have the faintest idea. Well, let's see, I do know that it's bigger than MIMI, and it's less than MIMIMI, and I do know how to figure it out if I have to, but I don't want to. I usually use MIMI, or MIMIMIMI, or just try around until it looks right.

MU

MU is useful when you want something to happen by chance. A spell that has many MU's in it makes a different shape each time it is cast, by chance.

MU is a random repeater. It could be worth MO, or MA, or ME or MI, and you never can tell which it will be. It's like throwing dice: each time it could be different. So MAMU could be 4 or 5 or 6 or 7 (anywhere in the second row). MUMA could be 1 or 5 or 9 or 13 (anywhere in the second column). MUMU could be anywhere between 0 and 15. And MUMUMUMU could be anywhere between 0 and 255.

For example, MUMUTUBILZIM does anywhere between 0 and 15 BIL's.

MX

MX lets you use a square of color to determine how many times something will repeat. MX repeats its word depending on the color last stepped on! That is, if the color last stepped on was KIKA, then MX is worth MIMA (13)! And strangely enough, MXMEMX would be worth 13*4*4 + 2*4 + 13 = 229. Remember, though, that on some computers repeaters can't get bigger than 255. Beyond 255, they would start over again at zero (like 256=0, 257=1, 258=2, and so on).

NU

NU repeats its word indefinitely. The word will not stop repeating until an IX or OX exits the word.

Pen Color

The pen color is the color used by BO to draw (and by GO to wash). When your spell is stopped, you can always tell what color your pen is by looking at the top square of the Matchbox.

You can change the pen color just by typing the name of the color. For example, KE NUBOIX makes a line in the color KE. The next page tells you the names of all the colors.

How many colors can Spellcaster draw? That depends on your computer (different computers have different numbers of colors). Of course, if you're not using a color TV or monitor, you won't see colors even if your computer can make them. Instead, you'll see different textures of dot patterns. Spellcaster comes with a spell named COLORS which will show you all the colors that Spellcaster uses on your computer. The tutorial will tell you all about the colors on your computer.

The names of the colors follow the same system as the repeaters (MO, MA, ME, MI). If you understand the names of the repeaters, you'll understand much better the names of the colors.

KO KA KE KI

If Spellcaster on your computer uses four colors, they will be named KO, KA, KE, and KI.

If Spellcaster on your computer uses sixteen colors, they will be named like this:

KOKO (KO)	KOKA (KA)	KOKE (KE)	KOKI (KI)
KAKO	KAKA	KAKE	KAKI
KEKO	KEKA	KEKE	KEKI
KIKO	KIKA	KIKE	KIKI

KOKO is the same as KO, KOKA is the same as KA, KOKE is the same as KE, and KOKI is the same as KI.

To change the pen color, you just type the name of the color you want. For example, if you want to draw a line in the color KEKA, you would type

KEKA NUBOIX

KU

KU is a random color. It could be KO, KA, KE or KI, and, like throwing dice, you never know what will come up. If you've got a KU in a loop (like NUKUSO), it may be different every time the loop repeats.

Likewise, KEKU is a random color that begins with KE (either KEKO, KEKA, KEKE or KEKI), and KUKE is a random color that ends with KE. KUKU could be any color between KOKO and KIKI.

AKA again

AKA changes the pen color to the next higher color. For example, if the pen color is KO, AKA changes it to KA. AKA changes KA to KE, KE to KI, and (if your computer has more than four colors), KI to KAKO. AKA changes the highest color to KO, the lowest.

The Color Last Stepped On

"The color last stepped on" is very important in Spellcaster: it can be used by a spell to decide which way the spell ought to go. "The color last stepped on" is the color of the screen square stepped on by the most recent BO, JO or GO. BO, JO and GO are the only syllables that "step". Of course, BO and GO change the color of the screen square. However, "the color last stepped on" is the color the square was, before it got changed.

The Matchbox

The Matchbox is that little box that's always on the drawing screen. Whenever the spell is stopped, the Matchbox shows two colors: the pen color (on top) and the color last stepped on (on the bottom). Sometimes it looks as if it only has one color in it: that's just because the other color is the same as the background.

Matched and Unmatched

Whenever the pen color matches the color last stepped on, the spell is "matched" (just then). Whenever the pen color is different from the color last stepped on, the spell is "unmatched".

If the two colors shown in the Matchbox are the same, the spell is matched. If they're different, it's unmatched.

KX

KX is useful if you want your pen to "pick up" the color of some square it has just stepped on. KX sets the pen color to match the color last stepped on. Immediately after a KX, a spell is guaranteed to be matched.

Exits

Exits (IX, OX and XEN) are the logical heart of Spellcaster. They let you make spells that behave differently depending on what they find on the screen. An exit forces its spell to make a decision: keep going as usual, or skip to something different.

How does the spell make the decision? By whether or not it is matched. A spell is matched if the color last stepped on is the same as its pen color (quickfind "The Color Last Stepped On" if you don't understand matching).

IX OX

IX and OX can be put anywhere in a word. More than one can be put in a word. They sometimes cause an exit. That means, they sometimes cause the rest of their word to be skipped, so that the spell continues with the following word.

For example, the words "BOBOBOIXBOBO RIAKA" would sometimes do 5 BO's before the RIAKA, but other times do only 3 BO's before the RIAKA, because the last two would get skipped.

IX does nothing, just lets the spell pass through it unchanged, if the spell is unmatched when it gets to the IX. But if the spell is matched, IX exits the word (the rest of the word is skipped over and the next word starts).

So BOBOBOIXBOBO would do only 3 BO's if the spell was matched by the end of the third BO.

OX is the opposite of IX. If the spell is matched when it gets to an OX, OX lets the spell pass through it. But if the spell is unmatched, OX causes the rest of the word it is in to be skipped, and the next word begins. So BOBOBOOXBOBO would do only 3 BO's if the spell was unmatched by the end of the third BO.

If a word has a repeater prefix, an IX or OX in the word might cause the repeating to quit before it has finished its usual number of repetitions. This is especially true for NU, because a NU word without an exit will never quit. For example, NUBOIXAKABO will only quit when the first BO happens to step on a square the same color as the pen color.

XEN

XEN is pronounced "ZEN". XEN is a prefix that must come even before repeater prefixes. For example, you can do XENMIMIBO but you can't do MIMIXENBO (try it, it won't let you type the XEN).

A XEN does an exit unless an IX or OX in the preceding word did an exit! Suppose your spell had the two words "MAMAIXBO XENRI" in it. There are two ways the spell might finish the MAMAIXBO: the MAMA (5) repeats might get all done, or the IX might cause an exit. In the first case, when the 5 repeats got finished without the IX ever doing its thing, the XEN would cause an exit (because no exit happened in the first word), so the RI would get skipped. In the second case, when the IX did an exit, the XEN wouldn't, so the RI would make the spell take a right. The word before a XEN must have at least one IX or OX in it, or the XEN word will always be skipped.

In the spell "IXTUJAMESZIM XENTUJOHNZIM", either JAMES or JOHN happens, and the other gets skipped. They never both happen. Why? Well, suppose the spell is matched when it gets to the IX. The IX will work, skipping over JAMES. So that means that the XEN will let JOHN happen. On the other hand, suppose the spell is unmatched when it gets to the IX. The IX won't exit, so JAMES will happen. But now the XEN will make the spell skip the JOHN.

Notice that these two spells do exactly the same thing, no matter what JAMES and JOHN do:

IXTUJAMESZIM XENTUJOHNZIM

OXTUJOHNZIM XENTUJAMESZIM

Why do they always do the same thing? Well, if either one of them starts matched, it will do JOHN, and not JAMES. And if either starts unmatched, it will do JAMES, and not JOHN. If you wanted it the other way around, you'd have to write it like either of these:

IXTUJOHNZIM XENTUJAMESZIM

OXTUJAMESZIM XENTUJOHNZIM

Both of these do the same thing, but their thing is the opposite of the first two spells.

Recursion

A spell can cast itself. For example, the spell

BOTUCATZIM AKABORIBOLI SPELLS TUCATZIM !

casts itself. When you first type it in, the computer will complain (as you type the first TUCATZIM) that "There are no spells named CAT", but that's all right, you're creating it! Go ahead and make CAT on your computer.

What happens when you cast CAT now, after it is defined? The computer looks up the definition of CAT, does the BO, and then says "hmm, I need to cast CAT again, but I'd better keep track of this first CAT, 'cause I'm not finished with it yet, so I'll call this second dude CAT2 or somethin'." Then it looks up the definition of CAT again, and does the BO, and then says "hmmmm, another CAT!?. O.K, that one's CAT3, and here goes..." Then it looks up the definition of CAT again, and does the BO, and...

Well, you see how things are going to go. It will just do BO's until it runs out of memory keeping track of all those CATs. The computer's got a lot of memory, it can keep track of hundreds of CATs, but there is a limit! Then the computer will print the message "Can't do it. Not enough room in memory." It didn't even get to the second word of the first CAT! Notice that even if it had a hundred times as much memory, the same thing would have happened.

A spell that casts itself (or a spell that casts another spell that... ends up casting the first spell) is called a "recursive" spell. What's the use of recursive spells if the computer always runs out of memory? Well, you can write recursive spells which don't run out of memory. How? Well, there has to be some way that sometimes the spell doesn't cast itself.

You've got to use an exit, IX or OX.

For instance, try this spell:

BOIXTUDOGIIM AKABORIBOLI SPELLS TUDOGIIM !

The only difference between DOG and CAT is the IX. Now when you cast DOG, the same thing's going to happen as with CAT, except that, perhaps on DOG143, the computer's going to do the 143'rd BO, which is finally going to step on the square that was colored by the first BO. DOG143 will be matched when it hits its IX. So it will skip the TUDOGIIM, and finally do an AKABORIBOLI. DOG143 will then be done, so the computer can now finish off DOG142 (doing its AKABORIBOLI), and then finish off DOG141, and DOG140, and so on all the way back to the first DOG, and be done!

When you write a recursive spell, you have to make sure that under some condition, the spell won't cast itself.

Sometimes, the easiest way to get something done is with a recursive spell. The FILL spell is a recursive spell.

Recursion

A spell can cast itself. For example, the spell

BOTOCATSIM ASARUBIBOLI SPELLS TUCATSIM !

casts itself. When you first type it in, the computer will complain (as you type the first TUCATSIM) that "There are no spells named CAT". But that's all right, you're creating it! Go ahead and make CAT on your computer.

What happens when you cast CAT now, after it is defined? The computer looks up the definition of CAT, does the BO, and then says "hmm. I need to cast CAT again, but I'd better keep track of this first CAT, 'cause I'm not finished with it yet, so I'll call this second dude CAT2 or somethin'." Then it looks up the definition of CAT again, does the BO, and then says "hmmmm, another CAT!?. O.K, that one's CAT3, and here goes..." Then it looks up the definition of CAT again, and does the BO, and...

Well, you see how things are going to go. It will just do BO's until it runs out of memory keeping track of all those CATs. The computer's got a lot of memory, it can keep track of hundreds of CATs, but there is a limit! Then the computer will print the message "Can't do it. Not enough room in memory." It didn't even get to the second word of the first CAT! Notice that even if it had a hundred times as much memory, the same thing would have happened.

Teleporters

Suppose you're a farmer driving tractor in a field. For some reason (supper?) you have to quit, and you want to pick up work later exactly where you left off. You'd need to remember where you quit, and what direction you were going, and whether you were disking or plowing or whatever.

Now suppose you wanted to stop the work of a spell, and later on have the spell pick up the work again at that place. You'd need to remember all these things:

Position of a Spell:

Location: where was the spell on the screen?
Direction: was the spell headed up, down, left or right?
Pen color: what color would it draw with if it did a BO?
Quiet or Buzzing: has a QUSH turned the BOs' clicks off?
Normal or Mirror: has a Y turned left to right and right to left?

That's what teleporters are for: remembering, and then later going back to, the position of a spell. (Notice there's one interesting thing that teleporters don't remember: the color last stepped on. I'll talk more about that in a bit.)

Read the next page for more about teleporters.

E FE

To make a teleporter remember the current position of a spell, you have to "expose" it, as if it took a picture. You can think of a teleporter as some kind of magic crystal. When you pull the crystal out of your pocket ("expose" it), it sees and remembers your position. When you expose a teleporter, it remembers the location, direction and pen color of the spell, and it remembers whether a QUSH has turned the BO clicking off, and whether RI and LI have been switched.

To make a teleporter take the spell back to the position the teleporter is remembering, you have to "felt" it (as if you had to reach into your pocket and rub the crystal with a felt cloth to make its magic work). When you "felt" a teleporter, it sets the location, direction and all that other stuff back to what they were when the teleporter was last exposed.

There are 26 teleporters, named A, B, C and so on through Z. To expose a teleporter, you type E followed by the name of the teleporter (for example, to expose teleporter A, you type EA). To "felt" a teleporter, you type FE followed by the name of the teleporter (for example, to "felt" teleporter A, you type FEA).

There's one thing a teleporter doesn't remember: the color last stepped on. Why? Well, that's so you can pass news from one teleporter place to another. Suppose you're working at teleporter A and just now stepped on a green square (so the color last stepped on is green). Then you can go to the place teleporter B is remembering, set the pen color to green, and start drawing in green, like this: FEBKXBO. The KX sets the pen color to be the color last stepped on (not the color you last stepped on when you were working at B, but the

color that was most recently stepped on by the spell, in this case the green back at A1.

Using Teleporters with Exits

KAKAEAKIKIRI MIMIBO FEAIXTUJAMESZIM XENTUJOHNZIM

When this spell runs into the IX, exactly what is the "color last stepped on", and what is the "pen color"? (It's important to know, otherwise you can't figure what the IX is going to do.) Let's start at the beginning. The KAKA sets the pen color. Then the EA exposes teleporter A. Let's call this place "position Abel." The pen color is then changed to KIKI. Then the spell takes a right and does fifteen (MIMI) BO's. Let's call the place where the spell is now, "position Baker." Then comes FEA, which sets the spell back to position Abel. That includes setting the pen color back to KAKA!

And then comes the IX. What is the color last stepped on? Well, it depends on what color the last BO stepped on at position Baker. Let's suppose you started this spell off by itself on an all black (KO) screen: the color last stepped on would be KO. Is the spell matched? No, the pen color is KAKA.

So on an all KO screen, this spell would always be unmatched by the time it got to the IX, so JAMES would happen. On a screen that already had enough KAKA drawn on it so that at position Baker the spell stepped in KAKA, JOHN would happen.

Teleporters and the Border

What happens if you "felt" a teleporter before you have exposed it? It takes you to its birthplace, its homeland, the border.

The border is a queer place. It's like a railroad track that goes all the way around the main screen. The track is only one little square wide. You can go straight ahead on the track (when you get to a corner you'll go around the corner, just doing straight BO's, JO's or GO's) or you can turn around (with a DI), but LI and RI have no effect at all. It's as if there were only two directions in the border, clockwise and counterclockwise. Even HI, in the border, doesn't make you go up, it makes you go clockwise (so that, for example, if you were way over on the right hand side of the border, a HI would actually turn you downwards).

When you're in the border, you can't turn and go into the main screen, and when you're in the main screen, you can't pierce through into the border. The only way to get from the main screen to the border, and vice versa, is with teleporters.

The "inborn" position of every teleporter is as follows:

location: the bottom left corner of the border screen.
direction: going up (clockwise).
pen color: Whatever color spells start with on your computer.
Quiet or Buzzing: buzzing.
Normal or Mirror: normal.

Once you're in the border, how do you get back? Easy, you just have to expose one teleporter in the main screen first, before you "felt" an unexposed one to go to the border. Then, when you want to go back, you just "felt" the one that's remembering the position in the main screen. For instance,

EA FEB MIMIMIMIBOAKA FEA MIMIMIMIBOAKA

Suppose you're in the border, and teleporter A is remembering a position back in the main screen, and you do BO FEA IXTUJOHNZIM XENTUJAMESZIM. The IX compares the color last stepped on (in the border!) to the current pen color (in the main screen!), to see if they match.

Why have a border? It's handy if you want to make some patterns that will govern spells that are drawing on the main screen. You don't want the main screen spells to overwrite the patterns that are governing them. The border is a nice, sheltered place for the governing patterns.

How do you make stuff in the border affect stuff in the main screen, or vice versa? The only way is with the color-last-stepped-on. For instance, suppose you had just stepped on a green square in the main screen. Then, if teleporter Z is in the border, FEZKX will set the pen color at Z in the border to be green.

Level Teleporters

When I talked about EA through EZ, and FEA through FEZ, I said that there were 26 teleporter stones. Actually, there are more, but the ones I'm about to tell you about are strange. Whenever a the computer

A. begins working on a spell that you are typing, or
B. comes to a TU...ZIM, or
C. comes to a PU...ZIM, or
D. comes to a WU...ZIM, or
E. comes to a VUZIM,

the computer snaps its fingers and plucks a new teleporter out of the air to use in the new level it's about to begin. What do I mean by level? Well, if spell A casts spell B, then spell B is one level below spell A. If spell A has a part in it that is between PU and ZIM, that part is one level below spell A. For example, if spell A is

BOBOTUJILZIMRIPU **MIMIBOTUWILZIMAKA** ZIMBORIBOLI

then A's level is BOBOTU..ZIMRIPU..ZIMBORIBOLI, JIL is one level below A, the MIMIBOTU..ZIMAKA is one level below A, and WIL is two levels below A. Any spell that JIL casts is two levels below A. Any spell that WIL casts is three levels below A, and so on. Every time you change levels, the printing changes from regular to boldface. This helps to see one level, but it makes it look like stuff two levels down is the same as stuff on the top level, which it isn't.

E' FE'

OK, so there's a special teleporter to use on each level. What's its name? It is called '. That's right, '. To "expose" a level teleporter, you type E', and to "felt" a level teleporter, you type FE'. For example, in this spell,

MIMIE'PU RIBOBOE'BOBOPE'RIBOBO ΣIMFE'JOJOJOJOAKA

there are actually two teleporters being used, because they're at different levels (even though they look as if they've got the same name). The following spell does exactly the same thing as the previous one:

MIMIEAPU RIBOBOEBBOBOPEBRIBOBO ΣIMPEAJOJOJOJOAKA

only it does it with two regular teleporters, A and B.

So why have level teleporters? Why not always use regular teleporters? Two reasons:

1. You might run out of regular teleporters (there are only 26 of them).
2. Recursion!

Recursion? Right. Let's change our recursive DOG spell (that we used when we discussed recursion) so it uses teleporters. I'm going to try twice, the first time using regular teleporters, the second time using level teleporters:

```
BOIXEZTUBAADOGZIM FEZAKABORIBOLI SPELLS TUBAADOGZIM !

BOIXE'TUGOODOGZIM FE'AKABORIBOLI SPELLS TUGOODOGZIM !
```

Try them out on your computer. Both of these spells are recursive, each casts itself. Each of them is going to go down many levels before it stops casting itself.

BAADOG exposes the same teleporter at each level, teleporter Z, so that when the computer finally starts coming back up through all those levels, doing the FEZAKABORIBOLI's, why they all teleport to the same place, the place remembered by the last exposure of teleporter Z. Not too interesting.

GOODOG exposes a different teleporter at each level of itself, so that when the computer finally starts coming back up through all those levels, doing the FE'AKABORIBOLI, each level teleports back to where it was before it cast the next level down. That can be very useful. Look again at how the definitions of DOG, BAADOG and GOODOG are different, and how they behave differently when you cast them.

Spells in the computer's memory, Spells on the disk

Some of the menu commands have to do with spells in the computer, and some of them have to do with files of spells on the disk. What's the difference? Well, whenever you define a spell, the computer remembers it in its memory. But turning the power off kind of blows the computer's mind. So if you want to turn your computer off for the day, tomorrow you'll have to type in all your spells again, unless you somehow keep a copy of your spells somewhere else.

That's where the disk comes in. You can command the computer to copy all the spells that are in its memory into a file on the disk. Turning the power off does not affect the disk. So tomorrow, when you turn your computer back on, all you do is command the computer to copy the disk file back into its memory, and you're all set.

You can even keep more than one file of spells on the disk, maybe one that you're working on, and one that a friend is working on.

Let's go through that again. Suppose the only spells in your computer's memory are the spells SALLY, COW, WILLIAM and WHAMO. You tell the computer to copy these into the file named MYSPELLS. Then the next day, you make up a spell called GEORGE, and then tell the computer to copy the spells from file MYSPELLS into memory. What happens? GEORGE gets wiped out, the only spells in memory now are SALLY, COW, WILLIAM and WHAMO. O.K, so you cool off, type GEORGE in again, change WHAMO to SUPERWHAMO, and then turn off your computer. The next day, you have the computer copy the spells from file MYSPELLS into its memory. What do you get? SALLY, COW, WILLIAM and WHAMO, because you forgot to save the changes and additions! Just because you changed WHAMO in memory didn't change it on the disk. Always remember you're dealing with two separate copies, and the only copies you can change are the ones in the computer's memory.

You might not want the spells from the tutorial spells file to clutter up a bunch of spells that you are making. Well, you can command the computer to erase them all (or use the Replace command to erase them one by one), so that you can start with a clean memory. There's no harm in erasing them from memory, because they're still on the disk.

D: Display the definition of a spell

The Display command is useful if you want to look at the definition of a spell, but you don't want to change it. If you want to do surgery on a spell, use Replace or Copy.

When you choose this command, the computer will ask you which spell you want displayed. Just type the name, hit ZIM to end the name, and the computer will display that spell.

R: Replace the definition of a spell

The Replace command is useful for erasing spells, or for changing them.

BEWARE! With this command it's easy to change, or even erase, wipe out, destroy, a spell that you've been working on for a long time! When you choose this command, the computer will ask you which spell you want to replace. Just type the name and hit ZIM to end the name. What happens next is this:

1. The computer erases the spell from its spell memory.
2. The computer puts the entire spell, minus the last keystroke (which is the final ZIM) into the keystroke stack, exactly as if you had typed it yourself and then hit the superleftarrow key.
3. The computer lets you start typing.

Let's do an example. Suppose you replace TUBILZIM. The computer will erase the spell from spell memory, put it in the keystroke stack, and you'll be left staring at a blinking B (since the first keystroke in BIL happens to be a B for BO). Now what?

1. Suppose you want to erase the spell. Just hit RETURN. Dead, gone, as if it had never been there.

2. Suppose you want to see the spell do its stuff. Hit superrightarrow (+), and the computer types (and draws)

BOBOBOBOBORIBOBOBOBORIBOBORIBOBORIBOBOBOBO SPELLS TUBIL

Everything's there except the final ZIM. Suppose you look at the spell and decide it's OK. All you do is hit the ZIM, and the spell is back in business exactly as if you'd just typed it in anew.

3. Suppose you want to change the spell's name to SAM. After hitting superrightarrow (+), just delete three keystrokes to back over the BIL, type SAM, hit ZIM, and it's done.

4. Suppose you want to change the spell itself. If there's a whole lot of changing to be done, maybe it's best to erase the spell (hit RETURN) and start over.

If the changes you want to make are near the beginning of the spell, then hit superleftarrow (-) to get you to the front of the spell, use rightarrow to move in to the place you want to change, make the change, and then use superrightarrow to finish typing it.

If the changes you want to make are near the end of the spell, hit superrightarrow and then use the leftarrow to back into the spell. Make your changes, use rightarrow to retype the end of the spell, and rename it.

Every time you use delete (/) or leftarrow, the whole spell is recast up to where your cursor is, and that can be a drag if the spell takes a long time. If you use the zapperleftarrow key (*), you'll be able to make your changes much faster. The zapperleftarrow key works like the superleftarrow key, but it also makes Spellcaster quit spellcasting, so you can work without waiting. The rub is that you can't see anymore exactly what effect your changes are having on the spell. If you hit the zapperleftarrow key a second time, it turns spellcasting back on, so you can see the results.

C: Copy the definition of a spell

The Copy command is useful if you want to make a new spell that is just a bit different from an old spell. It's easier to change a copy of the old spell than to type in a whole new spell.

If you choose this command, the computer will ask you for the name of the spell you want copied. Type the name of the spell, and hit ZIM to end the name.

The Copy command is very much like the Replace command. Take a look at what the Replace command does. The only difference between the two is that the Copy command skips the first step of the Replace command (where the Replace command deletes the spell from the spell memory). What difference does this make?

1. If you hit RETURN when you are working on a Copied spell, the original copy is still in the spell memory. If you hit return when you are working on a Replaced spell, the spell is gone, gone, gone.

2. If you hit superrightarrow when you are working on a Copied spell, and then hit % for ZIM, the computer will complain that there's already a spell by that name (of course, the original copy). When you copy and change a spell, you have to give it a new name.

L: List the names of all spells in memory

This command is useful if you've forgotten the name of a spell you want to use. The List command makes the computer list the names of all the spells in its memory (not the ones on the disk). There might be a whole lot. If there are more names than will fit on the screen, the computer will fill up the screen with names, and then pause to give you time to read them. While it's waiting, it displays the message "Hit L for more". When you hit L again, it will go on and display more names, until there are no more.

P: Print all spells in memory

The Print command is useful if you want a printed copy of all your spell definitions. This is especially handy when you're making a large system of spells that cast each other: sometimes it's nice to work on paper.

E: Erase all spells in memory

This command makes the computer erase all spells from its memory. It doesn't hurt the spells in disk files. Just in case you hit this command by mistake, the computer will check if you really mean it. It warns you about what it's going to do, and then asks you to type Y (for YES) if you really meant it. If you hit anything besides Y, it won't forget a thing.

O: Out (replace the disk file with a copy of the spells in memory)

This command makes a disk file that is a copy of all the spells in the computer's memory. It wipes out the old file (if there was one) and replaces it with a new file that has only those spells in it that are now in the computer's memory.

Now you might hit this command by mistake, and not want your file changed at all. Don't worry, the computer will warn you about what's going to happen, and ask you to type Y (for YES) to be sure you really meant it. If you type anything besides Y, the computer won't change your file.

I: In (replace the spells in memory with a copy of the disk file)

This command erases all spells from the computer's memory, and copies the spells from a disk file into the cleared memory.

Now you might have some new spells in memory that you've been working on, and hit this command by mistake. So, just like the Erase command, the computer warns you about what's going to happen, and asks you to type Y (for YES) to be sure you really meant it. If you type anything besides Y, the computer won't forget your spells.

A: Add in (add a copy of the disk file to the spells in memory)

This command is useful if you want to add some spells in one file to some spells in another file (or some new spells you've just typed). Use the In command to get the spells from the first file (or type in your new spells), then use the Use command to change the filename to the name of the second file, then use the Add command to add the spells from the second file. After all this, the computer's memory will contain all the spells from both files (so you could use the Out command to make a file that contained them all).

Suppose there was a spell named GEORGE in both files? The List command will show them both. If you cast TUGEORGE%IM, you'll cast the topmost one on the list. If you Display or Replace or Copy TUGEORGE%IM, you'll display, replace or copy the topmost one on the list. So you can easily change the name of the topmost one, and then you can get at them both.

U: Use a different disk file

This command lets you change the name of the file used by the Out, In and Add commands. When Spellcaster program starts, it sets the file name to SPELLS. To change the name, choose this command by hitting U, then type a file name.

Let's do an example. Suppose you want to make a file that has only your spells in it, and that will be called "MYSPELLS".

1. Use the Erase command to clear out all the spells that go along with the tutorial (ICHABOD and BIL and all).
2. Type a bunch of spells of your own.
3. Use the Use command to change the file name to MYSPELLS.
4. Use the Out command to create file MYSPELLS with a copy of all your spells in it.

A: Add in (add a copy of the disk file to the spells in memory)

This command is useful if you want to add some spells in one file to some spells in another file (or some new spells you've just typed). Use the In command to get the spells from the first file (or type in your new spells), then use the Use command to change the filename to the name of the second file, then use the Add command to add the spells from the second file. After all this, the computer's memory will contain all the spells from both files (so you could use the Out command to make a file that contained them all).

Suppose there was a spell named GROBGZ in both files? The List command will show them both. If you cast TOGEOBGEIM, you'll cast the topmost one on the list. If you Display or Replace or Copy TOGEOBGEIM, you'll display, replace or copy the topmost one on the list. So you can easily change the name of the topmost one, and then you can get at them both.

The Spellcaster Page Editor

The page editor is a program that lets you make your own tutorial pages. I used the page editor to making the Spellcaster tutorial. You can use it to tell people how to run your games (when you swap spell disks with friends), or to make your own user's group newsletter, or to make courseware for students to learn about computer programming, and so on. You can even market your own Spellcaster games or courseware, using the page editor to make a tutorial for your software.

Special Keys

The delete (/), rightdelete (\), leftarrow and rightarrow keys work just like they do in Spellcaster. For the superleftarrow and superrightarrow keys you have to use the uparrow and downarrow keys (if your computer has none, you have to use control-K and control-J). The other special keys (RETURN,!,?,<,>,+,-,*) do not behave in the special way they do in Spellcaster. Rather, they just behave like they would on a normal keyboard.

Page Conventions

If you want your pages to work right, there are a few things you'd best know. The easiest way for you to learn how to make a good page is to study the ones in the tutorial. Go ahead and use the page editor In command to bring a copy of the first page of the tutorial, P001, into the computer's memory.

File names for pages must begin with the prefix P, and are numbered from P000 to P999. You can edit them with prefixes other than P, but Spellcaster will only read ones whose prefix is P. I usually make them up with some other prefix, and then rename them all when I'm ready to go.

XXX

When you look at the tutorial's file P001, you'll notice it starts with XXX. All the pages do. That's just to reserve room for the page number, which depends on the file name. For instance, even if file P001 began with 486 (instead of XXX), it would still show up in Spellcaster as page 001. If you changed it's file name to P333 (without changing the XXX or anything else on the page at all), then it would show up in Spellcaster as page 333. This is handy: if you want to change the page numbers (maybe because you want to insert a page in the middle of a bunch of pages you've already made), all you have to do is rename the files (you don't have to edit the files at all).

Hints

Now look at the bottom line of the tutorial's first page. You see something with the page editor that you didn't from Spellcaster. That last line is a hint (as in "take a hint"), it's a bunch of keystrokes that are put into the keystroke stack whenever that page is read from Spellcaster.

Every page must end with exactly one hint. The hint should always start on a line of its own. A hint starts with @ and ends with @ (though these @'s are not put into the keystroke stack, just the stuff between them is).

The simplest hint is @@. That's really no hint at all. But every page must have a hint, so you use @@ when you don't want any keys put into the keystroke stack.

Writing hints is a little funny, because you have to pretend you're in Spellcaster. If you put in a string of keystrokes that won't make sense to Spellcaster, then your hint won't work right.

The best way to learn about writing hints is to study the hints used in the tutorial's pages.

Some Useful Spells Completely Explained

The SNIF spell looks for a matching square (a square the same color as the pen) one square ahead, or to the left, or to the right. The PARabola spells draw all kinds of curves. The FILL spell fills in an area of the screen with a solid color. This section explains how to cast these spells.

This section also takes these spells apart, and explains, step by step, how they work. Don't read about how these spells work until you have played a lot with all of the syllables of Spellcaster, and you're getting bored with simple spells. If you feel you want to move on to some more interesting (and difficult!) spellcasting, then you're ready for this section. SNIF is easier to understand than the PARABola spells, and FILL is hardest.

SNIF

SNIF is useful for following a line, like a bloodhound sniffing along a trail. If you start on a line, and do a SNIF before each step, the SNIF will automatically turn you in the right direction to follow the line.

How to use it:

To use the SNIF spell, you set the pen color to the color of the square you want SNIF to look for. Then do TUSNIFZIM. SNIF looks first straight ahead, then left, then right. If it finds a matching square, it turns in that direction, and quits matched. If it doesn't find a matching square, it quits unmatched. So TUSNIFZIMIX or TUSNIFZIMOX will exit (or not) depending on whether or not SNIF found a matching square. For instance, if you start on a line of the same color as your pen color, this spell

NUTUSNIFZIMOXBO

will follow the line to the end and then stop (though it can get confused if the line makes loops).

SNIF step by step:

EAJOFEAIXLIEAJOFEAIXDIEAJOFEA SPELLS TUSNIFZIM

EA	Remember the beginning position.
JO	Step forward.
FEA	Go back to beginning position.
IX	Skip all the rest of the spell if the first JO stepped on a matching square.
LI	Turn left.
EA	Remember the left-turned position (which is the same as the beginning position except you've turned left).
JO	Step forward.
FEA	Go back to the left-turned position.
IX	Skip all the rest of the spell if the second JO stepped on a matching square.
DI	Turn around.
EA	Remember the right-turned position (which is the same as the beginning position except you've turned right).
JO	Step forward.
FEA	Go back to the right-turned position.

Notice that if any of the JO's steps on a matched square, SNIF will end matched. If none of the JO's steps on a matched square, SNIF will end turned to the right, and umatched.

The PARabola spells

The PARabola spells draw curves. There are 5 spells in the PARabola family: PARBEGIN, PARIN, PAROUT, CURVIN and CURVOUT.

How to use:

PARBEGIN is always cast first, before you cast any of the others, to set up some stuff that the others depend on. PARIN does one step of an increasing curve (a curve that starts off nearly straight, but turns harder and harder as it goes along). PAROUT does one step of a decreasing curve (a curve that starts off curving real hard, but that gets straighter and straighter as it goes along). CURVIN makes one whole increasing curve by casting PARIN a number of times. CURVOUT makes one whole decreasing curve by casting PAROUT a number of times. To get the feel for these, try typing

```
TUPARBEGINZIMTUCURVINZIMTUCURVOUTZIM
MIMITUCURVINZIM
```

The PARabola family makes heavy use of teleporter W. Using chains of CURVIN and CURVOUT, separated by changes to teleporter W, results in strange snaky loops, repeated arches, and so on. To get a feel for this, take a look at the definitions of the spells HEAD, RING, BLOB and SINE.

If you want CURVIN and CURVOUT to make longer, lazier curves (or shorter, faster ones), add more BO's to PARBEGIN (or take some out).

How it works:

The general idea of CURVIN, for example, is to do

```
4 BO's straight ahead, then shift over one to the right, then do
3 BO's straight ahead, then shift over one to the right, then do
2 BO's straight ahead, then shift over one to the right, then do
1 BO   straight ahead, then shift over one to the right, and quit.
```

(CURVOUT works like CURVIN in reverse, first doing 1 BO straight, then two BO's, then three, then four, and then it quits.)

The Parabola spells

The Parabola spells draw curves. There are 5 spells in the PARABOLA family: PARBEGIN, PARIN, PAROUT, CURVIN and CUROUT.

How to use:

PARBEGIN is always cast first, before you cast any of the others, to set up some stuff that the others depend on. PARIN does one step of an increasing curve (a curve that starts off nearly straight, but turns harder and harder as it goes along). PAROUT does one step of a decreasing curve (a curve that starts off curving real hard, but that gets straighter and straighter as it goes along). CURVIN makes one whole increasing curve by casting PARIN a number of times. CUROUT makes one whole decreasing curve by casting PAROUT a number of times. To get the feel for these, try typing

```
[illegible]
[illegible]
```

The Parabola family makes heavy use of teleporter W. Using chains of CURVIN and CUROUT, separated by changes to teleporter W, results in strange snaky loops, repeated arches, and so on. To get a feel for this, take a look at the definitions of the spells SNAC, RING, BLOB and EGGS.

If you want CURVIN and CUROUT to make longer, lazier curves (or shorter, faster ones), add more GO's to PARBEGIN (or take some out).

PARBEGIN:

To start with, PARBEGIN lays down a strip of color over in the bottom left corner of the border. The length of this border strip depends on how many BO's there are in PARBEGIN. (You can change them if you want. With only three BO's, CURVIN would only count down from 3. With eight BO's, CURVIN would count down from 8.)

PARBEGIN step by step:

EWFE'BOBOBOBO SPELLS TUPARBEGINZIM (

EW	Make teleporter W remember a position in the main screen (you've got to expose a teleporter in the main screen, if you're ever going to get back from the border). The other spells in this family all use teleporter W to keep track of where to draw in the main screen.
FE'	Go to the border land, bottom left corner (where all teleporters are born).
BOBOBOBO	Lay down a strip of 4 squares of color.

PARIN, PAROUT, CURVIN and CURVOUT:

Teleporter Y is used to keep track of how many BO's you're going to go straight, and teleporter X is used to count those BO's. If teleporter Y is three BO's from the end of the border strip, you're working on doing 3 BO's straight ahead. The way you count three BO's is, you just position teleporter X at teleporter Y, and then advance teleporter X down the border strip, doing one BO in the main screen for every step you take on the strip. When teleporter X gets to the end of the strip, you do a RIJOLI in the main screen (to shift over one to the right), then you advance the Y teleporter one step (so that next time you'll only do 2 BO's straight ahead before you shift), set teleporter X at teleporter Y and start over.

PARIN does one step of this process (PAROUT does the same thing, only in reverse). When teleporter Y gets to the end of the strip, though, PARIN has got to start Y all over again at the beginning of the strip. This is the only time that PARIN ends unmatched (normally PARIN ends matched).

So CURVIN's job is pretty easy, all it does is cast PARIN until PARIN goes unmatched. In fact, I might as well give you the definitions for CURVIN and CURVOUT right here, there's nothing more to say about them:

CURVIN and CURVOUT definitions:

```
NUTUPARINZIMOX SPELLS TUCURVINZIM !
NUTUPAROUTZIMOX SPELLS TUCURVOUTZIM !
```

And now for PARIN and PAROUT. Remember, they use teleporter W to keep track of where they are drawing in the main screen.

PARIN step by step:

```
FEXJOEXOXFEWBOEWKX XENFEWRIJOLIEWFEYJOEYEXIXFE'EYEXKXAKA
SPELLS TUPARINZIM !
```

FEXJOEX	Advance teleporter X one step.
OX	If you've run off the end of the strip, skip to the following XEN word.
FEWBOEW	Do a BO in the main screen.
KX	End this word matched.
XEN	Do this word when you've run off the end of the strip.
FEWRIJOLIEW	Do a RIJOLI in the main screen.
FEYJOEY	Advance teleporter Y one step.
EX	Set teleporter X to be the same as teleporter Y.
IX	If Y hasn't run off the end of the strip, quit matched.
FE'EYEX	Set X and Y back to the beginning.
KXAKA	Quit unmatched.

Notice the trick PARIN uses to make sure it quits matched or unmatched. To guarantee that it's matched, it does KX, to guarantee that it's unmatched, it does KXAKA. There are three places where PARIN could quit: after the first word, after the IX in the second word, and at the end of the second word. Only in the last place does PARIN quit unmatched.

PAROUT is just like PARIN, only it has to work backwards:

```
FEXDIJODIEXJOOXFEWBOEWKX XENFEWLIJORIEWFEYJOEYEXIXFE'EYEXKXAKA
SPELLS TUPAROUTZIM !
```

FEXDIJODIEX	Move teleporter X back one step.
JOOX	If you're past the corner (the beginning), skip to the following XEN word.
FEWBOEW	Do a BO in the main screen.
KX	Quit matched.
XEN	Do this word when you've moved back past the beginning.
FEWLIJORIEW	Do a LIJORI on the main screen.
FEYJOEY	Advance teleporter Y one step.
EX	Set teleporter X to be the same as Y.
IX	If Y hasn't run off the end of the strip, quit matched.
FE'EYEX	Move X and Y back to the beginning.
KXAKA	Quit unmatched.

And now for PARIN and PAROUT. Remember, they use teleporter W to keep track of where they are drawing in the main screen.

PARIN step by step:

```
FEXJOEXOXFEMBOEWEX XENFEMBIJOLIEWFEYJOEYEXIXFE'EYEXEXAKA
SPELLS TUPARINXIM :
```

FEXJOEX	Advance teleporter X one step.
OX	If you've run off the end of the strip, skip to the following XEN word.
FEMBOEW	Do a BO in the main screen.
EX	End this word matched.
XEN	Do this word when you've run off the end of the strip.
FEMBIJOLIEW	Do a BIJOLI in the main screen.
FEYJOEY	Advance teleporter Y one step.
YX	Set teleporter X to be the same as teleporter Y.
IX	If Y hasn't run off the end of the strip, quit matched.
FE'EYEX	Set X and Y back to the beginning.
EXAKA	Quit unmatched.

The FILL spell

There are two kinds of fill spell one might imagine:

Type 1 fills a region of constant color surrounded by an outline made of one or many colors (other than the region's color).

Type 2 fills a region that may already contain many colors that is surrounded by an outline of constant color (that is different from any of the colors in the region).

Something has to be constant. Either (Type 1) the region is all one color (to begin with), or (Type 2) the outline is all one color. There's got to be some way to figure out where the boundaries of the region are! The FILL spell I am about to explain is of Type 1. You could make up a Type 2 FILL spell, in fact it would be nice to have both.

How to use it:

Suppose you have just outlined a region that you want filled with a certain color. You need to position your spell on the outline of the region, turned into the region, with pen color set to the color you want to fill the region. Then do TUFILLZ. When FILL is done, the position will be unchanged.

Here's an example that draws a small square and fills it in.

```
MAMOPU MAMOBO RI ZIM BORIKETUFILLZIM
```

How it works:

There are three spells in the FILL system: FILL, FILA and FILB. FILL sets up initial conditions, and casts FILA. FILA casts FILB, and FILB casts FILA (that's recursion). FILA and FILB really do the work.

FILA goes straight ahead, casting FILB to right and left. FILA is like the spine of a feather, casting FILB's like fronds to both sides. If you watch the spell in action you can see this happening. FILA stops when it gets to a wall (the outline).

FILB also goes straight until it runs into a wall. Only, it feels to either side of it for a wall, and if it happens to pass alongside of a hole in the wall leading to right or left, it casts a new FILA into the hole.

To begin with, FILL is positioned on the outline of the region, turned into the region, with pen color set to the "fill" color. Call the color of the region (before FILL is called) the "region" color. FILA and FILB need to remember these two colors somehow. FILA and FILB use the "region" color as their pen color, because they're always looking for squares that match that color. They know the "fill" color by looking one step behind them: they always presume that the square one step behind them has already been filled in, so they look there to find out what the "fill" color is.

The FILL spell:

There are two kinds of fill spell one might imagine:

Type 1 fills a region of constant color surrounded by an outline made of one or many colors (other than the region's color).

Type 2 fills a region that may already contain many colors that is surrounded by an outline of constant color (that is different from any of the colors in the region).

Something has to be constant. Either (Type 1) the region is all one color (to begin with), or (Type 2) the outline is all one color. There's got to be some way to figure out where the boundaries of the region are! The FILL spell I am about to explain is of Type 1. You could make up a Type 2 FILL spell, in fact it would be nice to have both.

How to use it:

Suppose you have just outlined a region that you want filled with a certain color. You need to position your spell on the outline of the region, turned into the region, with pen color set to the color you want to fill the region. Then do TUFILLZ. When FILL is done, the position will be unchanged.

Here's an example that draws a small square and fills it in.

```
MAMOPU MAMOBO RI IIM BOSIKETUFILLEIM
```

Like I said, the FILL spell just sets things up for FILA and FILB. FILL has three jobs to do:

1. Remember the starting position, so that when FILA and FILB are done, FILL can go back to the starting position. (Why? Because it's nicer for whoever casts FILL to be returned to their position, instead of ending up who knows where.)

2. Cast FILA just right: when you cast FILA, the pen color must be the region color, and the square just stepped on must already be filled in with the fill color.

3. When FILA is done, go back to the starting position.

FILL step by step:

```
E'BOKXTUFILAZIMFE' SPELLS TUFILLZIM !

E'          Remember the starting position.
BO          Fill in one square with the fill color.
KX          Set the pen color to the region color.
TUFILAZIM   Cast FILA.
FE'         Return to the starting position.
```

Tactics of FILA and FILB:

Tactic 1: FILA and FILB operate with their pen color set to the region color, because they're always looking for squares that match the region color, in order to fill them in. So how do they know what the fill color is? They look behind them: both FILA and FILB assume that the square just behind them has already been filled in (by themselves or, for the very first square, by FILL). To color one square in the region with the fill color, here's the little song and dance that FILA and FILB do:

```
DIJOKXDIBO

DIJO        Turn around and step on the square behind you.
KX          Set the pen color to the color you just stepped on
            (which hopefully is the fill color).
DIBO        Turn back in the direction you were going and
            fill in the square you were on.
```

Tactic 2: FILB marches steadily ahead, but if it notices that it's skimming along a wall, it keeps a lookout for a hole in the wall. If it finds a hole, it casts a FILA into the hole. How does FILB detect a hole? Well, if it goes past two squares, and the first one is wall (i.e., not in the region), and the second one is region colored, then a hole has opened up.

Like I said, the FILL spell just sets things up for FILA and FILB. FILL has three jobs to do:

1. Remember the starting position, so that when FILA and FILB are done, FILL can go back to the starting position. (Why? Because it's nicer for whoever casts FILL to be returned to their position, instead of ending up who knows where.)

2. Cast FILA just right: when you cast FILA, the pen color must be the region color, and the square just stepped on must already be filled in with the fill color.

3. When FILA is done, go back to the starting position.

FILL step by step:

```
:'HOME:TUFILA:IMPS' SPELLS TUFILL2IN :

:'          Remember the starting position.
BG          Fill in one square with the fill color.
SE          Set the pen color to the region color.
TUFILA2IN   Cast FILA.
PE'         Return to the starting position.
```

FILA step by step:

```
NUE'RITUFILBZIMFE'LITUFILBZIMFE'JOE'OXDIJOKXDIBOFE'
SPELLS TUFILAZIM !
```

```
NU                  Repeat.
E'RITUFILBZIM       Cast FILB to the right.
FE'LITUFILBZIM      Cast FILB to the left.
FE'JOE'             Advance position one square.
OX                  Quit repeating if the new square isn't in the region.
DIJOKXDIBO          Fill in the new square (see Tactic 1).
FE'                 Set the pen color back to the region color.
```

FILB step by step:

```
NUJOE'OXDIJOKXDIBOFE'PU E'LIJOOXLIJOIXFE'LITUFILAZIM
FE'RIJOOXRIJOIXFE'RITUFILAZIM ZIMFE' SPELLS TUFILBZIM !

NU              Repeat
JOE'            Move position forward one square.
OX              Quit repeating if the new square isn't in the region.
DIJOKXDIBO      Fill in the new square (see Tactic 1).
FE'             Set the pen color back to the region color.
PU
  E'LIJOOX        Test for hole to the left.
  LIJOIX          Test for wall behind and to the left (see Tactic 2).
  FE'LITUFILAZIM Center and cast FILA into a left hole.

  FE'RIJOOX       Test for hole to the right.
  RIJOIX          Test for wall behind and to the right (see Tactic 2).
  FE'RITUFILAZIM Center and cast FILA into a right hole.
ZIM
FE'             Return to position.
```

What makes FILA and FILB interesting is that they mix recursion with NU repetition (iteration). It is much easier to make a FILL system that only uses recursion, but then it is limited to relatively small regions. If the machine has to do one recursion for each square that it colors, it runs out of memory (remembering all the recursions) if you give it a region that is large.

FILA step by step:

```
NUE'RITUFILBZIMFE'LITUFILBZIMFE'JUE'OXDIJOHXDIBOFE'
SPELLS TUFILASIM :

NU                  Repeat.
E'RITUFILBZIM       Cast FILB to the right.
FE'LITUFILBZIM      Cast FILB to the left.
FE'JUE'             Advance position one square.
OX                  Quit repeating if the new square isn't in the region.
DIJOHXDIBO          Fill in the new square (see Tactic 1).
FE'                 Set the pen color back to the region color.
```

Spellcaster Syntax

definition := spell SPELLS TUspellnameZIM !

A spellname is zero or more alphnums.

An alphnum is an uppercase ABCDEFGHIJKLMNOPQRSTUVWXY, or space, or 0-9, or possibly others.

A spell is zero or more words, separated by spaces.

word := [XEN][repeater]atomstring

repeater := NU ! mrepeaterstring

An mrepeaterstring is zero or more mrepeaters.

mrepeater := MO ! MA ! ME ! MI ! MU ! MX

An atomstring is zero or more atoms.

atom := AKA ! BO ! DI ! Etelename ! FEtelename ! GO ! HI ! IX !
JO ! KA ! KE ! KI ! KO ! KU ! KX ! LI ! OX ! PU spell ZIM !
QUSH ! RI ! TUspellnameZIM ! VUZIM ! WU ! Y

A telename is an uppercase A-Z, or a '.

INDEX

A
 AKA 15,39
 Add in (menu) 68
activity change keys 18

B
 BO 10
BAADDOG spell 58
border 10
   and teleporters 54

C
 Copy (menu) 64
casting a spell 25,28
changing a spell 63
changing the file name
   (see use)
counters (see repeaters)
color 37-40
color last stepped on 40
command menu 59
computer literacy 7
CURVIN, CURVOUT
   (see PARabola spells)

D
 DI 14
 Display (menu) 61
DOG spell 49
delete key 22
deleting a spell
   (see Replace)

E
 E 52
 E' 57
 Erase (menu) 65
editing spells
   (see Replace,
   Keystroke stack)
editor 71
exits 43-45
   with teleporters 53
exposing a teleporter 52

F
 FE 52
 FE' 57
felting a teleporter 52
files 59
   (also see In, Out,
    Add in, Use)
 FILL spell 84-89

G
GO 12
GOGDDOG spell 58

H
HI 14

I
IX 44
In (menu) 67
if, if then 40,43-47

J
JO 11

K
KA KE KI KO 38
KU 39
KX 41
keystroke stack 19-23

L
LI 14
List (menu) 65
left arrow key 22
levels (with teleporters) 56
logic 40, 43-47

M
MA ME MI MO 34
MU 36
MX 36
matchbox 40
matching 40
menu 59
memory 59,60
mode (see activity
change keys)

N
NU 36
noise 66
numbers (see repeaters)

O
OX 44
Out (menu) 67

P
PU 28,29
Print (menu) 65
page editor 71
page conventions 72
PARabola spells 78-83
pen color 37-39
position of a spell 51
process state
(see position of a spell)
programming 5

Q
 QUSH 15

R
 RI 14
 Replace (menu) 61
recursion 48,49,58
repeaters 33,34
right arrow key 22
right delete key 22

S
 SPELLS 26 (also see
     spell, below)
 Switch sound (menu) 66
SNIF spell 76
sound 66 (also see BO, GO)
special keys 17
spell
 what is a spell 3
 casting 25,28
 defining 25-27
 deleting (see Replace)
 naming (see defining)
stack, see keystroke stack
syntax 91
switch 66
superleftarrow key 21,23
superrightarrow key 22,23

T
 TU 28
 Turn page (menu) 66
teleporters 51-58
 with exits 53
 and the border 54
 and levels 56

U
 Use (menu) 69

V
 VUZIM 28,31

W
 WU 28, 30
washing 12

X
 XEN 46
 XXX 72

Y
 Y 15

Z
 ZIM (see TU, PU, WU)
zapperleftarrow key 21,23

G
GO 12
GOSUB spell 58

H
HI 14

I
IK 44
In (menu) 67
if, if then 40,43-47

J
JO 11

K
KA KE KI KO 38
KO 39
KX 41
keystroke stack 19-23

L
LI 14
List (menu) 65
left arrow key 22
levels (with teleporters) 50
logic 40, 43-47

M
MA ME MI MO 14
MU 38
MX 38
matchbox 40
matching 40
menu 59
memory 59,60
mode (see activity
change keys)

N
NU 38
noise 66
numbers (see repeaters)

O
OX 44
Out (menu) 67

P
PU 28,29
Print (menu) 65
page editor 71
page conventions 72
PANabola spells 78-83
pen color 37-39
position of a spell 51
process state
(see position of a spell)
programming 5